SATISFIABILITY problem is NP-complete.

NP-complete:

a) SATISFIABILITY is in NP

b) For every language L in NP, L <=P SATISFIABILITY

(Cook’s Theorem)

SATISFIABILITY: n variables, a giant logic proposition on those n variables. Is there an assignment to the n variables that makes the proposition true?

Let L be a problem in NP, that means we have a nondeterministic (1-tape) machine M that decides L. Build a giant table nk x nk..

Each row represents the current state of the tape of the machine.

From row I to row i+1 we do one step of machine M.

We created a giant logic statement with lots of variables. O(n2k)

a) Each entry of the table has a valid symbol.

b) Each row is a valid step of the Turing machine M

c) The first row has just input x on the tape, M is in state q1 and the head is at the first symbol

d) qaccept shows up in the table in some row

x is in L if and only if the SATISAFIABILITY instance that we created has a solution.

(←) If there is an assignment to the variables that makes the proposition true: a) we start with x on the tape in state q1, and M will achieve state qacceot.  Then x is accepted by M, x is in L.

If there is a valid assignment to the variables, then there is some table that shows an accepting computation, then M will accept input x.

Example: CNF-SAT

A logical proposition in conjunctive normal form. We have n variables, m clauses, each clause is made of of literals: x, x We want to assign T/F to the variables so that each clause has at least on true literal.

(x OR (not y) OR z OR (not w) OR a) AND ((NOT x) OR y OR w OR b) AND …

Example: 3-SAT : A CNF-SAT problem where each clause has exactly 3 literals.

Prove: 3-SAT is NP-complete.

Lemma: A language A is NP-complete if:

a) A is in NP

b) B <=P A where B is some NP-complete language.

Proof: Let L be an language in NP. B is NP-complete, L <= B. B <= A. Therefore L <= A. x is in L, f(x) is in B, g(f(x)) is in A. (f circ g)

CNF-SAT <=P 3-SAT.

(Prove 3-SAT is in NP. Guess an assignment to the variables of 3-SAT and in time linear in the number of clauses, verify that each clause has one true literal.)

Given m clauses and n variables that is an instance to CNF-SAT. We have to create a 3-SAT instance (we want the size of the 3-SAT instance to be a polynomial of the size of the CNF-SAT instance), we want there to be a solution to the 3-SAT instance if and only if there is a solution to the CNF-SAT instance.

CNF-SAT: 3-SAT

(x) (x,b,c),(x,(NOT b),c), (x,b,(NOT c)), (x, (NOT b), (NOT c))

(x,y) (x,y,a),(x,y,a)

(x,y,z) (x,y,z)

…

(x,y,z,w,v,u) (x,y,d),((NOT d), z, e), ((NOT e), w, f), ((NOT f), v, u)

Is this a polynomial time reduction: How many clauses in the 3-SAT instance, how many variables?

M’ clauses = 4(# 1 clauses) + 2 \* (# 2-clauses) + (# 3-clauses) + (k-2)(# k-clauses, k > 3)

n’ variables: n + 2(# 1clauses) + (# 2 clauses) + (k-3)(# k clauses, k >= 3)

CLIQUE is NP-complete. (3-SAT <= CLIQUE)

Vertex Cover = {<G,k>} G is a graph, k is a number, there exists a subset of at most k vertices such that every edge is connected to one of these k vertices.

Vertex Cover is NP-complete

a) Vertex Cover is in NP. Guess a set of k vertices. Check in linear in the number of edges if every edge has one of these k vertices as an end point.

b) 3-SAT <= Vertex Cover.

This is a gadget proof. We need to create “gadgets” to represent assignments and clauses.

We are given an instance of 3-SAT. We are given n variables and m clauses. We want to find an assignment to the variables that satisfy the clause.

We have to create a G and a k for vertex cover.

G will contain 2n + 3m vertices and a bunch of edges.

(x,y,z), ((NOT x), y, a), ((NOT y), (NOT z), b)

Need k = n + 2m

If we can cover all edges with exactly k vertices, then there exists an assignment to the variables that satisfies each of the clauses.

SUBSET SUM is NP-complete

SUBSET SUM : n numbers (integers) and a target number T. Is there a subset of the integers that sums to T?

(the integers are all in binary)

Vertex Cover <= SUBSET SUM

Given a graph G and a number k, we need to create a set of numbers and a T.

For each vertex, create a number:

a is connected to e1 e3 and e4

b is connected to e1 and e2

v e1 e2 e3 e4 ...

vertex a: 1 1 0 1 1

vertex b: 1 1 1 0 0

edge e1: 0 1 0 0 0

edge e2: 0 0 1 0 0

T: k 2 2 2 2 2 2